# The OXO Data Format

Mount

earth@mount.build

**Abstract**

The OXO data format is a way of representing arbitrary graph and binary data. This paper introduces a rough specification of the OXO data format, and demonstrates usage by example.

## Class

The OXO data exchange format ("Oh-show") is a class of binary data format to encode arbitrary graph data fields of bits of data. The goal is to register the media type "application/oxo" and the file extension ".oxo".

The goal of this specification is only to define the syntax and base semantics of valid OXO shifts. It intentionally does not define how a valid OXO text might be internalized into the data structures of a programming language. There are many possible semantics that could be applied to the OXO syntax and many ways that a OXO text can be processed or mapped by a programming language. Meaningful interchange of information using OXO requires agreement among the involved parties on the specific semantics to be applied. Defining specific semantic interpretations of OXO is potentially a topic for other specifications. Similarly, language mappings of OXO can also be independently specified.

## Match

A conforming OXO text is a sequence of bits that strictly conforms to the OXO grammar defined by this specification. A conforming processor of OXO shifts should not accept any inputs that are not conforming OXO shifts. A conforming processor may impose semantic restrictions that limit the set of conforming OXO shifts that it will process.

# Shift

An OXO *build* is composed entirely of a *shift* sequence. Each shift is composed of 4 parts, the *drive*, followed by the *scale*, then the *count*, and finally the *chain*. There are 4 drives that capture the structure of graph records and bits. They are *stack*, which is the construction of a record with fields. The *field*, which are elements of a stack. The *block* or binary data itself. And *chain* for grouping records or data. The *drive* is a 2-bit sequence which is defined as follows:

```
stack = xx
field = xo
block = ox
chain = oo
```

The *scale* is a sequence that is a multiple of 2. It is used to specify how many bits we need to use to encode the *count*. It is necessary so that the count can grow arbitrarily large, although for now it is practically limited to 32-bits. For example, the scale would look like this:

```
2 bits  = xx
4 bits  = xo
8 bits  = ox
16 bits = ooxx
32 bits = ooxo
64 bits = ooox
...
```

Finally, the *count* is a *power* of two, but specifies the individual number of bits that follow. Even though it is allowed to be 32 bits, the maximum number of bits it should specify is limited in size to $2^{28}$ or roughly 268 billion bits, or roughly 33 megabytes. This is at least for the first version of OXO. This doesn't mean the size of the file is limited to only $2^{28}$ bits, but that an individual field is limited to that. After the count, the *chain* is the actual individual bits, up to the size of the count. They are padded to 2-bit chunks. A single chain will not be longer than $2^{28}$ bits. A complete *shift* for a hypothetical class number 1, field number 1, with a value being the 8-bit sequence `xoxoxoxo`, would look like this:

We introduce here a brief example semantics to show how this might be parsed. First, we assume that the first stack bits point to a valid class object. This class object is assumed to have a number of fields which are unchanging. This is followed by one or more field bit sequences, which act as an index of the field in the class. That is, it is the position of the field within the class. This information can be used to parse the fields into the appropriate place in some external runtime datastore. Now that we have fields, they then point to either a chain or a trace (address).

The address is either direct bits, or another stack, and the process continues. In this way, we can construct records with semantic meaning into a graph database. Some of these records can also be actions, which could be used for computation. That is how OXO binaries could be used to store and share arbitrary information and computation.

The record structure of a shift is as follows:

```
class build
  field trick, class count
  field hatch, class hatch
  field shift, chain shift

class hatch
  field start, class count
  field crown, class count
  field front, class count

class shift
  field scale, class count
  field count, class count
  field block, chain block
```

The *build* is a class within the global class system with ID 0. In its base form the trick is always 5, and it is defined as part of a stack. So the shift sequence for the build trick is as follows, with the magic bits always at the 16th position:

# Grammar

This is a rough BNF grammar.

```
build = shift*
shift = drive scale count count(chain)
drive = xx / xo / ox / oo
scale = xx / xo / ox / ooxx / ooxo ...
count = 2(chain) / 4(chain) / 8(chain) / 16(chain) / 32(chain) ...
chain = block block
block = [ox]
```